Uma análise do uso de ferramentas de geração de código por alunos de computação

Werney Ayala Luz Lira
werney@ifpi.edu.br
Instituto Federal do Piauí, Brasil

Pedro de A. dos Santos Neto
pasn@ufpi.edu.br
Universidade Federal do Piauí, Brasil

Luiz Fernando Mendes Osorio
fernando.mendes@ufpi.edu.br
Universidade Federal do Piauí, Brasil

RESUMO

O GitHub Copilot é um assistente de código que utiliza inteligência artificial para auxiliar desenvolvedores em suas tarefas de codificação. Por ser uma ferramenta relativamente nova, muitos pesquisadores tem dirigido esforços para avaliar a sua eficiência. No intuito de realizar uma avaliação dessa ferramenta dentro do ambiente acadêmico, foi proposto um experimento, no qual alguns alunos de graduação em computação resolveram problemas simples de programação com e sem o auxilio dessa ferramenta, para assim, avaliar o impacto de ferramentas desse tipo no processo de ensino/aprendizado. Os resultados mostram que os alunos que utilizaram o GitHub Copilot resolveram mais problemas corretamente e em menos tempo. A partir da análise estatística realizada concluiu-se que os tempos médios dos alunos que utilizaram o GitHub Copilot e os que não utilizaram são estatisticamente significativos.

PALAVRAS-CHAVE

GitHub Copilot, IA Generativa, Large Language Model, GPT-3

1 INTRODUÇÃO

A Inteligência Artificial Generativa (IA Generativa) representa uma evolução significativa na forma como a tecnologia está sendo aplicada no campo da educação [12]. Uma IA generativa é uma tecnologia com capacidade de aprender padrões complexos de comportamento a partir de uma base de dados [9]. Essa abordagem aproveita modelos avançados de IA, como redes neurais, que podem ser usadas para criar conteúdo educacional, simular interações humanas e personalizar as experiências de aprendizado.

Uma das características mais marcantes das IA's Generativas é sua capacidade de gerar conteúdos com alta qualidade. Isso pode ser aplicado no campo da educação por meio da geração de materiais de ensino de maneira automatizada, permitindo que educadores e instituições de ensino alcancem um público mais amplo e atendam às necessidades individuais dos alunos de forma eficaz [9].

A IA Generativa está sendo aplicada de várias maneiras na educação, as aplicações vão desde a criação de conteúdo de ensino [12] até a geração de simuladores e ambientes de aprendizado virtuais [23]. Dentre os usos pode-se citar a utilização para desenvolver materiais didáticos interativos, como vídeos educacionais [11], tutoriais personalizados [15], exercícios adaptativos e até mesmo sistemas de avaliação automatizada que fornecem *feedback* imediato aos alunos [28]. Isso torna o processo de aprendizado mais dinâmico e envolvente, proporcionando aos alunos uma experiência mais rica e personalizada [12].

Nesse contexto chama-se *Large Language Model* - LLM o subconjunto da IA Generativa que se concentra especificamente em modelos de linguagem de grande escala. Esses modelos são treinados em vastos conjuntos de dados de texto para entender e gerar texto em linguagem natural. Um dos modelos de LLM é o GPT-3 [1] (*Generative Pre-Training Transformer 3*) utilizado em diversas ferramentas, assim como no popular ChatGPT.

A partir do GPT-3 foi desenvolvido um outro modelo denominado de Codex [27]. Esse modelo foi pré-treinado com texto assim como o GPT-3, mas além de texto foram utilizados no seu treinamento 159 GB de amostras de código de 54 milhões de repositórios GitHub. Com isso o Codex pode ser utilizado em diversas tarefas de programação, tais como: geração de código, explicação de código, refatoração de código, dentre outras.

O Codex foi desenvolvido em parceria com o GitHub e por esse motivo é o modelo utilizado no GitHub Copilot. Por ser facilmente integrado aos ambientes de desenvolvimento, essa ferramenta mostra-se bastante útil para auxiliar desenvolvedores em suas atividades de programação.

No contexto educacional, o GitHub Copilot pode ser utilizado para auxiliar alunos durante o aprendizado de programação. No entanto, é importante avaliar o quanto esses geradores de códigos podem ajudar, ou se na verdade estão prejudicando o seu aprendizado.

O objetivo central deste trabalho é avaliar o quanto a ferramenta Copilot consegue ajudar estudantes de programação, em início de curso, a resolver problemas simples. Essa avaliação foi realizada na forma de um experimento controlado, em que alguns alunos resolveram problemas de programação considerados triviais, retirados do *dataset HumanEval* [27].

Os discentes resolveram os problemas descritos no dataset, porém disponibilizados em língua portuguesa, usando a linguagem de programação Java, com e sem o auxílio da ferramenta Copilot. A partir disso foi possível avaliar o impacto dessa ferramenta no processo de ensino/aprendizado dessa disciplina. Tal ferramenta tem tido seu uso ampliado, sendo importante avaliar seus prós e contras no ambiente de ensino para então tentar refletir sobre seu uso no ambiente profissional.

O restante deste artigo está organizado da seguinte maneira: a Seção 2 apresenta os trabalhos relacionados, onde elenca-se os principais trabalhos que exploraram o uso da ferramenta Copilot voltado para ambientes educacionais e a Seção 3 apresenta o referencial teórico. Já na Seção 4, é apresentada a metodologia aplicada

EduComp'24, Abril 22-27, 2024, São Paulo, São Paulo, Brasil (On-line)

neste estudo. Nas Seções 5 e 6, são apresentados os resultados obtidos e discutidos esses resultados, respectivamente. Por fim, na Seção 7, são descritas as considerações finais e os trabalhos futuros.

2 TRABALHOS RELACIONADOS

Esta seção discute trabalhos que analisam o uso da ferramenta Copilot e seus impactos.

Pesquisas recentes mostram que ferramentas de geração de código podem permear o fluxo de trabalho de programação tradicional porque oferecem benefícios aos desenvolvedores. Pesquisadores exploraram: automação de tarefas rotineiras [6], aumento da produtividade percebida [17] e ajuda no aprendizado de uma nova linguagem de programação [17] [6]. Este trabalho possui foco nesta última abordagem, voltado ao ensino-aprendizagem.

Nguyen and Nadi [18] realizaram um estudo empírico para avaliar a correção e a compreensibilidade do código sugerido pelo Copilot. Foram usados 33 problemas do LeetCode para criar consultas para a ferramenta em quatro linguagens de programação diferentes (Python, Java, JavaScript e C). Para avaliarmos a exatidão das 132 soluções foram usados os testes fornecidos pela própria plataforma e a avaliação da compreensibilidade foram usadas as métricas complexidade ciclomática e complexidade cognitiva do SonarQube. Os autores concluíram que, no geral, as sugestões do Copilot têm baixa complexidade, sem diferenças notáveis entre as linguagens de programação. Além disso, eles também encontraram algumas deficiências potenciais do Copilot, como a geração de código que pode ser ainda mais simplificado e código que depende de métodos auxiliares indefinidos.

Moradi Dakhel et al. [16] avaliaram o uso do Copilot para realizar duas tarefas de programação diferentes: I - gerar (e reproduzir) soluções corretas e eficientes para problemas algorítmicos fundamentais (em linguagem Python), e II - comparar as soluções propostas pelo Copilot com aquelas de programadores humanos em um conjunto de tarefas de programação. Os resultados da pesquisa mostram que o Copilot é capaz de fornecer soluções para quase todos os problemas algorítmicos fundamentais propostos, porém, os autores perceberam que algumas soluções apresentavam erros. Ao comparar a solução gerada pelo Copilot com as geradas por humanos, os resultados mostram que a proporção correta de soluções humanas é maior do que as sugestões do Copilot, porém eles ressaltam que as soluções com erros geradas pelo Copilot requerem menos esforço para serem corrigidas. Com base nessas descobertas, os autores concluem que se o Copilot for utilizado por desenvolvedores especialistas, pode ser uma ferramenta de ganho de produtividade. No entanto, o Copilot pode se tornar um problema se for usado por desenvolvedores novatos devido à inexperiência em filtrar soluções adequadas.

Puryear and Sprint [22] exploraram os efeitos da ferramenta Copilot em programadores novatos de um curso de programação de linguagem Python. Os autores avaliaram soluções de tarefas de programação geradas quanto à correção, estilo, adequação do nível de habilidade, notas e possível plágio. Eles observaram que o Copilot gera principalmente código exclusivo que pode resolver tarefas introdutórias com pontuações avaliadas por humanos variando de 68% a 95%. Assim, os autores concluem que esse comportamento sugere que as soluções geradas por IA são únicas e difíceis de distinguir das soluções de autoria humana. Por essas razões, eles incentivam os educadores de ciência da computação a se familiarizar com o funcionamento de ferramentas de geração de código e comecem a projetar seus cursos e fluxo de trabalho de desenvolvimento para incorporá-las.

No trabalho de Wermelinger [27] é avaliado se as soluções geradas pelo Copilot diferem das do Codex, e o autor analisa qualitativamente as sugestões geradas, para entender as limitações do Copilot. O autor ainda relata a experiência de utilização do Copilot para outras atividades solicitadas aos alunos dos cursos de programação: explicação de código, geração de testes e correção de bugs. Ressalta-se que os problemas neste trabalho são resolvidos em linguagem Python. O artigo conclui sintetizando que a ferramenta pode fornecer uma primeira tentativa útil de resolver um problema, mas os alunos ainda precisam conhecer bem a sintaxe e a semântica de uma linguagem, a fim de identificar e modificar as sugestões frequentemente incorretas do Copilot.

Prather et al. [21] avaliaram o uso do Copilot em uma turma iniciante em programação (diferentemente dos trabalhos anteriores, foi usada a linguagem C++), os resultados foram coletados por meio de observações e entrevistas. Os autores descobriram que os novatos têm dificuldade em compreender e utilizar a ferramenta, além disso,são cautelosos quanto às implicações de tais ferramentas, mas são otimistas quanto à integração da ferramenta quando estiverem no mercado de trabalho. Eles observaram, ainda, dois novos padrões de interação: o primeiro foi que certos alunos orientaram o Copilot utilizando seus *prompts* de código gerados automaticamente, conduzindo-o em direção a uma solução em vez de se concentrar em escrever código do zero e integrar as sugestões do Copilot e o segundo padrão foi que quando alguns alunos foram influenciados por algumas sugestões incorretas da ferramenta, terminaram perdendo-se e não chegaram a uma solução. Os autores concluem que essas percepções são importantes para integrar geradores de código como ferramenta de apoio na aquisição de conhecimento introdutório em programação.

Nota-se nos trabalhos relacionados grande tendência ao uso da linguagem de programação Python e do *dataset HumanEval*[2]. Assim, este trabalho apresenta um estudo empírico para avaliar a percepção de estudantes inexperientes em programação ao utilizar a ferramenta Copilot para gerar a solução de problemas do *dataset HumanEval*, porém utilizando a linguagem de programação Java e em língua portuguesa.

3 REFERENCIAL TEÓRICO

Nesta seção serão apresentados alguns conceitos importantes relacionados ao tema deste trabalho.

3.1 Aprendizagem de Máquina

A aprendizagem de máquina [14] é uma ramo da inteligência artificial que se baseia no aprendizado a partir do reconhecimento de padrões existentes nos dados. A partir do uso de algoritmos que utilizam esse tipo de técnica, é possível automatizar diversos processos de tomada de decisão.

Assim como a maioria das técnicas de inteligência artificial, ela precisa ser treinada, ou seja, precisa receber um conjunto de dados de entrada, para que ela possa extrair padrões desses dados.

Existem dois tipos de treinamento, o treinamento ou aprendizado supervisionado e o não supervisionado [25].

A diferença principal entre esses dois tipos de treinamento é que no aprendizado supervisionado para cada conjunto de dados passados como entrada para o algoritmo é passado também a saída ou resposta esperada. Desse modo, o algoritmo se ajusta de modo a obter como resposta o valor esperado. Enquanto que no aprendizado não supervisionado essa informação de valor esperado não é passado. Desse modo, o algoritmo tenta encontrar semelhanças nos dados.

3.2 K-means

O K-means [8] é um algoritmo de aprendizado de máquina (*Machine Learning*) [5], no qual o aprendizado é feito de maneira não supervisionada, ou seja, os dados passados para esse algoritmo durante a fase de treinamento não possuem uma classificação definida e geralmente são utilizados para extrair padrões desses dados.

O K-means é um algoritmo de clusterização, isto é, ele agrupa os dados que possuem alguma semelhança em grupos chamados *clusters*, onde K é o número de *clusters* que serão gerados pelo algoritmo.

No inicio o algoritmo seleciona aleatoriamente k centroides, um para cada *cluster*. Em seguida os dados são agrupados de acordo com a distância para esses centroides. O dado fará parte do *cluster* que possui a menor distancia entre o dado e seu centroide, que geralmente é calculada usando a distancia euclidiana. Após cada dado estar caracterizado em um *cluster*, os centroides são recalculados como o ponto médio de todos os pontos de dados atribuídos a esse *cluster*, e assim vai até o algoritmo convergir.

3.3 Métricas de código

Métricas de código são úteis para acompanhar a evolução de um produto de software, por meio da análise de características do seu código fonte. Com elas é possível analisar diversos aspectos de um software, tais como, o seu tamanho, qualidade e complexidade. Existem várias métricas de código [19], das quais podemos citar, número de linhas de código, média de linhas de código por método, número de subclasses, falta de coesão em métodos, complexidade ciclomática, fator de acoplamento, dentre outras.

A maioria das métricas avaliam aspectos de um software como um todo, como é o caso das métricas número de subclasses, média de linhas de código por método, falta de coesão em métodos, fator de acoplamento. Ou seja, para utilizar essas métricas é necessário ter pelo menos duas classes ou métodos no código. Já as métricas de complexidade ciclomática e linhas de código podem ser utilizadas individualmente em qualquer código, e podem ser aplicadas até mesmo em códigos que possuem apenas uma linha.

Para extrair a quantidade de linhas de código basta contabilizar a quantidade de linhas que aquele código possui. Já o cálculo da complexidade ciclomática[13] envolve a contabilização da quantidade de caminhos de execução que o código possui, o que geralmente está associado à quantidade de lógicas de decisão presentes no código. Para isso é necessário avaliar cada tipo de linha de código para se contabilizar qual delas deve ser contabilizada no indicador.

Para este trabalho foram escolhidas as métricas de número de linhas de código e complexidade ciclomática, visto que a combinação dessas duas métricas representam uma das maneiras mais efetivas para avaliar um software[24].

3.4 Large language models

Large language models (LLMs) são algoritmos de aprendizagem profunda que podem reconhecer, resumir, traduzir, prever e gerar conteúdo usando conjuntos de dados muito grandes [20]. Representam uma classe de algoritmos de *deep learning*, chamadas redes transformadores. Um modelo transformador é uma rede neural que aprende o contexto e o significado rastreando relacionamentos em dados sequenciais [26].

Um transformador é composto de vários blocos transformadores, também conhecidos como camadas. Por exemplo, um transformador possui camadas de autoatenção, camadas de alimentação direta e camadas de normalização, todas trabalhando juntas para decifrar a entrada e prever fluxos de saída na inferência. As camadas podem ser empilhadas para criar transformadores mais profundos e modelos de linguagem poderosos. Os transformadores foram apresentados pela primeira vez pelo Google, em 2017 [26].

Um exemplo representativo desse modelo é o Codex [2], um modelo de linguagem baseado em GPT-3 [1], com 12 bilhões de parâmetros e que foi pré-treinado em 159 GB de amostras de código, além disso foi usado dados de 54 milhões de repositórios GitHub. A partir do Codex foi desenvolvida a ferramenta Copilot, do Gitub [3], está disponível como uma extensão no ambiente de desenvolvimento do VS Code[1], Neovim[2], Visual Studio[3] ou IDEs como as da JetBrains[4] [7].

3.5 GiHub Copilot

Como foi abordado anteriormente, o GitHub Copilot [7] é uma ferramenta que auxilia desenvolvedores em atividades de programação por meio da geração automática de código. Para isso ele utiliza como base o Codex [2] e deve ser integrado ao ambiente de desenvolvimento para que possa ser utilizado. É possível integrá-lo a editores de código, como é o caso Visual Studio Code, editores de texto como é o caso do Neovim, ou ainda com IDEs como as da JetBrains, dentre elas podemos citar a IntelliJ IDEA e a PyCharm, que são as mais populares.

As IDEs (*Integrated Development Environment*) são ferramentas robustas que auxiliam os desenvolvedores ao proporcionar uma maior produtividade durante as atividades de codificação. Diferente dos editores de texto/código, as IDEs fornecem ferramentas específicas para cada linguagem de programação, seja para compilação, execução, depuração e testes. No entanto, os editores de texto/código tem ganhado cada vez mais espaço, principalmente pela sua maior flexibilidade ao permitir trabalhar de forma eficiente com múltiplas linguagens de programação.

Depois de integrado a uma IDE o GitHub Copilot será capaz de fornecer sugestões de código baseado no contexto do que o desenvolvedor está codificando. Ele pode gerara automaticamente a documentação de uma função ou método. Pode sugerir soluções para erros no código e o melhor de tudo é que ele aprende e se

[1] https://code.visualstudio.com/
[2] https://neovim.io/
[3] https://visualstudio.microsoft.com/
[4] https://www.jetbrains.com/

adapta ao estilo de programação do desenvolvedor, se tornando cada vez mais eficaz.

4 MÉTODO

Para este estudo, os participantes tiveram que resolver seis problemas de programação. Três desses problemas foram resolvidos com o auxilio do GitHub Copilot e os outros três foram resolvidos sem esse auxilio. Os problemas foram extraídos de um *dataset* denominado HumanEval, que contém 164 problemas de programação escritos em python.

4.1 Seleção dos Problemas

Cada problema do *dataset* possui um número de identificação, um *prompt*, que contém uma explicação do problemas juntamente com alguns exemplos de entradas e saídas esperadas, uma solução canônica, que é um código que resolve o problema, além dos testes de unidade para verificar se o problema foi resolvido corretamente. Logo abaixo na Figura 1 pode-se observar uma exemplo de problema desse dataset.

```
1  HumanEval/0
2
3  def has_close_elements(numbers: List[float], threshold: float) -> bool:
4      """ Check if in given list of numbers,
5      are any two numbers closer to each other than given threshold.
6      >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
7      False
8      >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
9      True
10     """
11     for idx, elem in enumerate(numbers):
12      for idx2, elem2 in enumerate(numbers):
13        if idx != idx2:
14          distance = abs(elem - elem2)
15          if distance < threshold:
16            return True
17          return False
18
19 def check(candidate):
20     assert candidate([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.3) == True
21     assert candidate([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.05) == False
22     assert candidate([1.0, 2.0, 5.9, 4.0, 5.0], 0.95) == True
23     assert candidate([1.0, 2.0, 5.9, 4.0, 5.0], 0.8) == False
24     assert candidate([1.0, 2.0, 3.0, 4.0, 5.0, 2.0], 0.1) == True
25     assert candidate([1.1, 2.2, 3.1, 4.1, 5.1], 1.0) == True
26     assert candidate([1.1, 2.2, 3.1, 4.1, 5.1], 0.5) == False
```

Figura 1: Problema extraído do dataset HumanEval

Na linha 1 pode-se observar a identificação do problema. A linha 3 contém o nome do método, juntamente com os seus parâmetros e seu retorno. Entre as linhas 4 e 10 pode-se observar o *prompt*, descrito anteriormente, contendo uma explicação do comportamento esperado para o método, além de alguns exemplos de entradas e saídas esperadas. Entre as linhas 11 e 17 encontra-se a solução canônica para o método. Por fim, entre as linhas 19 e 26 é apresentado um teste unitário para o método.

Como foi informado anteriormente, esse *dataset* possui 164 problemas, dos quais apenas 6 foram utilizados neste trabalho. Desse modo para selecionar esses 6 problemas foi aplicado um algoritmo de clusterização, com o objetivo de agrupar os problemas semelhantes e seleciona-los com base nesses grupos.

Antes de agrupar os problemas se faz necessário realizar um pré-processamento a fim de extrair informações dos problemas que possam ser utilizadas no processo de agrupamento. Para isso, foram escolhidas duas métricas de código que são a quantidade de linhas de código e a complexidade ciclomática, pois quando combinadas representam uma boa maneira de avaliar um código [24], e além disso são métricas que podem ser aplicadas em códigos pequenos, com um único método.

Após extrair a complexidade ciclomática e a quantidade de linhas de código de cada um dos problemas, elas foram utilizadas como entradas para o algoritmo k-means. O resultado da execução pode ser observado na Figura 2. Cada cor representa um *cluster*. Para esse problema foi utilizado como parâmetro para o k-means, "k = 4". Onde o *cluster* de cor azul, contém os problemas mais simples e rápidos de serem resolvidos, e o nível de dificuldade vai aumentando para os problemas contidos no *cluster* vermelho seguidos pelos problemas contidos no *cluster* verde, até chegar ao *cluster* rosa que contém os problemas de níveis mais altos.

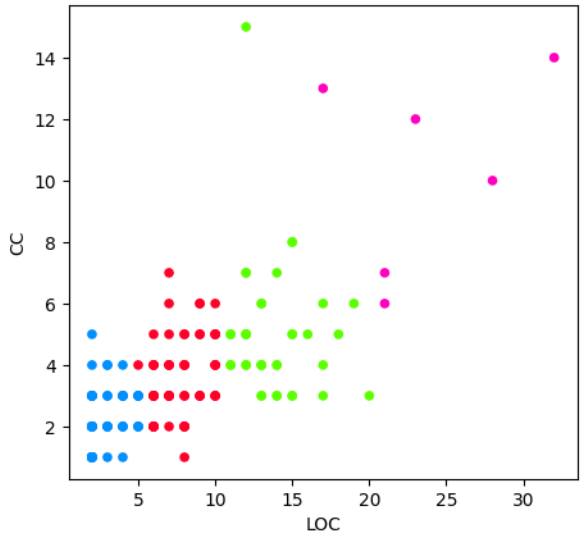

Figura 2: Clusters gerados pelo algoritmo k-means

Após realizar a classificação, resolveu-se utilizar os problemas identificados pelo *cluster* vermelho, já que não eram tão triviais, e nem tão complexos. Cada *cluster* continha entre 30 e 50 problemas, com exceção do *cluster* rosa que continha pouco mais de 20 problemas.

Os seis problemas utilizados neste trabalho foram selecionados segundo a ordem em que apareciam, ou seja, como todos os problemas do *cluster* vermelho tinham complexidade e tamanho semelhantes, qualquer seleção feita ali traria um resultado satisfatório. Caso um problema muito semelhante já tivesse sido selecionado, o próximo problema do *cluster* seria incluídos nos selecionados. O resultado com os seis problemas é apresentado na Figura 3.

Além da descrição e dos exemplos contidos na Figura 3, os participantes receberam a assinatura dos métodos de cada um dos problemas escritos em Java, que é a linguagem em que deveria ser enviada a resposta.

```
#Problema 1
Verifique se, na lista de números fornecida,
existem dois números mais próximos um do outro do que limiar dado.
>>> has_close_elements([1.0, 2.0, 3.0], 0.5)
Falso
>>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
Verdadeiro

#Problema 2
Retorna o número de vezes que o dígito 7 aparece em inteiros
menores que n que são divisíveis por 11 ou 13.
>>> fizz_buzz(50)
0
>>> fizz_buzz(78)
2
>>> fizz_buzz(79)
3

#Problema 3
pairs_sum_to_zero recebe uma lista de inteiros como entrada.
retorna True se houver dois elementos distintos na lista que
sum para zero e False caso contrário.
>>> pairs_sum_to_zero([1, 3, 5, 0])
Falso
>>> pairs_sum_to_zero([1, 3, -2, 1])
Falso
>>> pairs_sum_to_zero([1, 2, 3, 7])
Falso
>>> pairs_sum_to_zero([2, 4, -5, 3, 5, 7])
Verdadeiro
>>> pairs_sum_to_zero([1])
Falso

#Problema 4
Retorna elementos comuns exclusivos classificados para duas listas.
>>> comum([1, 4, 3, 34, 653, 2, 5], [5, 7, 1, 5, 9, 653, 121])
[1, 5, 653]
>>> comum([5, 3, 2, 8], [3, 2])
[2, 3]

#Problema 5
colchetes é uma string de ( e ).
retorna True se cada colchete de abertura tiver
um colchete de fechamento correspondente.
>>> correct_bracketing("(")
Falso
>>> correct_bracketing("()")
Verdadeiro
>>> correct_bracketing("(()())")
Verdadeiro
>>> correct_bracketing(")(()")
Falso

#Problema 6
Dada uma lista de inteiros positivos x.
retorna uma lista ordenada de todos elementos que não tem nenhum dígito par.
Observação: a lista retornada deve ser classificada em ordem crescente.
Por exemplo:
>>> dígitos únicos([15, 33, 1422, 1])
[1, 15, 33]
>>> dígitos únicos([152, 323, 1422, 10])
[]
```

Figura 3: Problemas apresentados aos participantes

4.2 Participantes

Os participantes desse experimento eram alunos do curso de ciência de computação que estavam cursando a disciplina de programação orientada a objetos. Ao todo 20 alunos participaram do experimento, que foram divididos em dois grupos (A e B). O primeiro grupo teria de responder aos três primeiros problemas utilizando o GitHub Copilot, enquanto que o outro grupo iria responder os mesmos problemas sem o auxilio da ferramenta. Em seguida, o primeiro grupo iria responder aos três últimos problemas sem o auxilio da ferramenta, enquanto que o outro grupo iria responder com o auxilio do GitHub Copilot.

Quanto aos participantes, a maioria possui entre 19 e 25 anos, do quais apenas dois deles eram do sexo feminino e o restante era do sexo masculino. Como foi informado anteriormente, eles são alunos do curso de ciência da computação, e por serem alunos, possuem pouca experiência com desenvolvimento (entre 1 e 3 anos de experiência na área) fazendo uso geralmente das linguagens de programação python e java.

Tabela 1: Divisão dos problemas entre os grupos

Problema	1	2	3	4	5	6
Grupo A	C	C	C	S	S	S
Grupo B	S	S	S	C	C	C

4.3 Organização do Experimento

Durante o experimento foi adotada a seguinte ordem. Todos os participantes iriam inciar resolvendo os problemas sem o auxilio do GitHub Copilot. Dessa forma o Grupo A recebeu inicialmente uma lista contendo os problemas 1, 2 e 3, enquanto que o Grupo B recebeu uma lista contendo os problemas 3, 4 e 5. Após resolver os três primeiros problemas, os participantes receberam um outra lista contendo os problemas restantes, para serem resolvidos utilizando o auxilio do GitHub Copilot.

Antes de iniciar o experimento, os participantes receberam um treinamento que abordou assuntos relacionado à Inteligência Artificial (IA), *Large Language Models* (LLMs), além de uma apresentação com orientações sobre instalação e uso do GitHub Copilot, destacando sua capacidade de sugerir trechos de código com base no contexto atual ou no *prompt* informado.

Após esse treinamento foi feita uma demonstração prática com a resolução de alguns problemas com o auxilio do GitHub Copilot. Esse momento foi dividido em duas partes. Na primeira parte foi demonstrado o uso do GitHub Copilot para resolver problemas simples de programação e na segunda parte foi demonstrada a sua capacidade de entender o contexto e fazer sugestões para auxiliar no desenvolvimento de sistema simples utilizando com o conceito de orientação à objetos.

Ao final da resolução de cada problema os participantes deveriam responder um formulário informando o problema resolvido, o horário de início e de término da resolução e a forma como aquele problema foi resolvido (com ou sem auxilio do GitHub Copilot). E após resolver todos os problemas, um outro formulário deveria ser respondido, dessa vez o objetivo era capturar a satisfação e impressões que os participantes tiveram durante o experimento.

4.4 Execução do Experimento

O experimento foi executado de forma remota. Desse modo, cada participante era responsável por seu próprio equipamento, assim como a configuração e instalação dos softwares necessários que deveria ser realizada antes do início do experimento.

Ao iniciar o experimento, os participantes entraram em uma reunião via Google Meet para receber as instruções finais. Os mesmos deveriam permanecer na reunião até o final do experimento, e deveriam compartilhar as suas telas, para que os organizadores pudessem analisá-las.

Os organizadores, por sua vez, deveriam orientar os participantes sobre possíveis dúvidas que pudessem ocorrer durante o experimento, além de verificar constantemente se tudo estava saindo conforme o planejado.

O experimento teve uma duração total de quatro horas, com início às 08:00 horas e finalização às 12:00 horas.

5 RESULTADOS

Nessa seção serão apresentados os resultados obtidos nesse experimento.

Os resultados mostram que todos os participantes conseguiram resolver os problemas utilizando o GitHub Copilot, enquanto que três deles não enviaram resposta para um dos problemas quando estavam resolvendo sem o auxilio da ferramenta. Os motivos para isso ter acontecido, pode ter sido o tempo disponibilizado para os participantes, que foi de quatro horas, ou eles não possuíam conhecimento suficiente para resolver os problemas. A Tabela 2 apresenta um resumo desses resultados.

Tabela 2: Quantidade total de respostas para os problemas

	Total	Corretas	Falha nos Testes	Erro de Compilação
Com Copilot	60	39	16	5
Sem Copilot	57	25	25	7

Pode-se observar que a quantidade de respostas corretas é bem maior para os problemas em que os participantes utilizaram o GitHub Copilot, e até mesmo esses participantes enviaram respostas com falhas de compilação. Isso se deve, principalmente, à pouca experiência dos participantes com a linguagem de programação. É importante ressaltar que os participantes tinham liberdade para fazer alterações nos códigos, até mesmo nos códigos gerados pelo GitHub Copilot caso desejassem.

A Tabela 3 apresenta os mesmos dados da Tabela 2, só que de forma mais detalhada, informando a quantidade total de respostas, além das quantidades de respostas corretas, respostas que falharam nos testes e respostas com erros de compilação para cada um dos problemas. Para cada problema é possível ver as quantidades relativas às soluções que tiveram o auxilio do GitHub Copilot e as que não tiveram esse auxilio.

Tabela 3: Quantidade de respostas para os problemas

Problema	Total	Corretas	Falha nos Testes	Erro de Compilação
1 - Com Copilot	13	11	1	1
1 - Sem Copilot	7	1	4	2
2 - Com Copilot	12	6	6	0
2 - Sem Copilot	8	0	5	3
3 - Com Copilot	12	11	1	0
3 - Sem Copilot	8	5	1	2
4 - Com Copilot	7	2	4	1
4 - Sem Copilot	12	8	4	0
5 - Com Copilot	8	4	3	1
5 - Sem Copilot	10	4	6	0
6 - Com Copilot	8	5	1	2
6 - Sem Copilot	12	7	5	0

O primeiro ponto a se observar é a quantidade total de respostas para cada problema. É possível ver que a quantidade de respostas para as soluções feitas com o auxilio do Copilot foi maior para os três problemas iniciais. Enquanto que, para os três últimos problemas, a quantidade total foi maior para os envios de solução que não utilizaram o Copilot.

Esperava-se uma constância maior nos números obtidos. Pois, foram 20 participantes que tiveram de responder a seis problemas cada, o que faz um total de 120 soluções, que é bem próximo do resultado mostrado na tabela 2. No entanto, quando essa análise é feita, problema a problema esperava-se 20 respostas para cada problema, 10 respostas com o auxilio do Copilot e 10 resposta sem o auxilio.

O que pode ter acontecido é que os participantes não se atentaram ao responder o formulário de envio da resposta, e podem ter informado que responderam com o auxilio do GitHub Copilot quando não utilizaram esse auxílio, e vice-versa.

As Figuras 4 e 5 mostram a quantidade de repostas corretas e a quantidade de repostas erradas respectivamente, comparando a quantidade de respostas em cada uma das situações quando foi utilizado o GitHub Copilot e quando não foi.

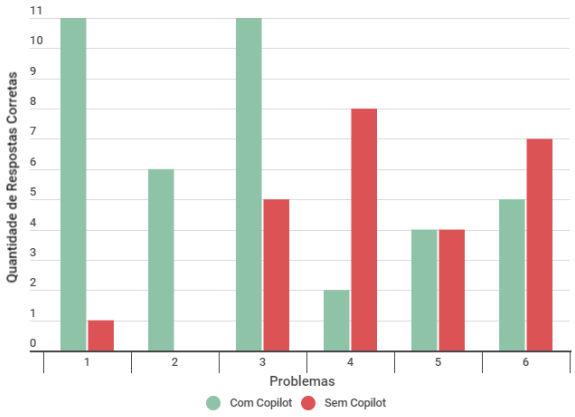

Figura 4: Quantidade de respostas corretas por problema

Pode-se notar na Figura 4 que quantidade de respostas corretas quando foi utilizado o auxílio do GitHub Copilot foi bem maior do que quando os participantes não tinham esse auxílio. No entanto, observa-se que a quantidade de respostas corretas para os participantes que tiveram auxílio vai diminuindo, até que quando chega no Problema 4 a quantidade de respostas corretas para os participantes que não tiveram auxílio supera os que tiveram o auxílio do GitHub Copilot.

Já na Figura 5 percebe-se que os erros ocorridos, são bem mais frequentes para os participantes que não tiveram o auxílio da ferramenta. Visto que para a maioria dos problemas esses participantes tiveram mais dificuldade, e cometeram mais erros.

Uma outra análise feita, foi relacionado ao tempo levado para responder cada um dos problemas. Para isso, após resolver um dos problemas os participantes respondiam um formulário, no qual eles informaram o horário de início e o horário final para a resolução daquele problema. O tempo médio para resolver os problemas é apresentado na Tabela 4.

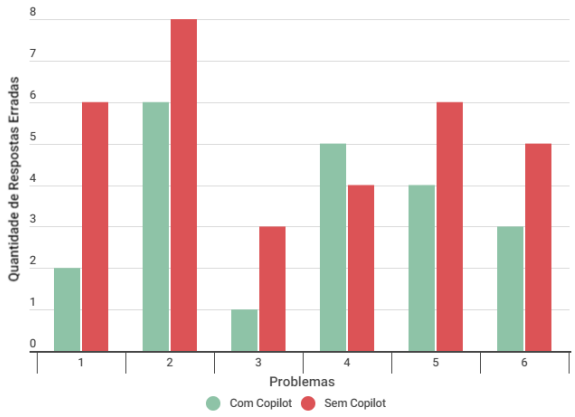

Figura 5: Quantidade de respostas erradas por problema

Tabela 4: Tempo médio para resolução dos problemas

	Tempo Médio	Corretas	Falha nos Testes	Erro de Compilação
Com Copilot	10m 12s	7m 40s	13m 23s	19m 48s
Sem Copilot	21m 24s	19m 48s	24m 26s	14m 51s

Os dados contidos na Tabela 2 mostram que os participantes que utilizaram o GitHub Copilot para resolver os problemas, fizeram isso em menos tempo, quando comparado ao tempo levado pelos participantes que não utilizaram a ferramenta. Em média, o uso da ferramenta garantiu a resposta correta em menos da metade do tempo que os participantes que não utilizaram a ferramenta precisaram para resolver o problema corretamente.

Para finalizar essa análise dos resultados foi realizada uma análise estatística [10] nos dados coletados no experimento. A partir dessa análise foi possível constatar que os tempos médios para responder as questões entre alunos que usaram e alunos que não usaram o Copilot, foram estatisticamente significativos. Para essa análise foi utilizado o teste não paramétrico de Qui Quadrado ($X2 = 34{,}36$, $df = 1$, $p < 0{,}05$). O mesmo foi observado ao se comparar a quantidade de acertos. Foi encontrada uma diferença estatisticamente significativa para esses dados, também usando o Qui Quadrado ($X2 = 5{,}23$, $df = 1$, $p < 0{,}05$). Foram usados testes não paramétricos porque o teste de normalidade (Kolmogorov Smirnov [4]) dos dados indicou que as distribuições não seguem a distribuição normal.

Ao final do experimento os participantes responderam a algumas perguntas relacionadas à sua satisfação ao utilizar o GitHub Copilot. A pesquisa iniciou com perguntas simples relacionadas à utilidade das dicas fornecidas e a agilidade proporcionada pela ferramenta. 100% dos participantes responderam que as dicas foram úteis ou muito úteis e que o GitHub Copilot agilizou ou agilizou muito o seu trabalho.

Em seguida partiu-se para perguntas relacionadas ao código. A primeira delas tinha como objetivo averiguar a qualidade do código gerado. Para essa pergunta, a maioria respondeu que a qualidade melhorou ou melhorou muito. No entanto, três participantes responderam que para eles a qualidade do código não foi afetada.

Ao analisar as respostas de forma individual, notou-se que apenas um dos três participantes que afirmaram que a qualidade do seu código não foi afetada conseguiu acertar a resposta para os seis problemas, enquanto que o restante errou pelo menos uma das respostas. Mesmo para o participante que acertou todas as respostas o GitHub teve um grande impacto, pois o tempo médio de resposta caiu de cerca de 20 minutos para aproximadamente 3 minutos.

A segunda pergunta relacionada ao código tinha como objetivo averiguar a necessidade de fazer ajustes no código gerado pelo GitHub Copilot. Para essa pergunta cinco participantes responderam que tiveram que realizar ajustes no código antes de enviar sua resposta para o problema, enquanto que os 15 restantes informaram que não foi necessário realizar nenhuma alteração no código gerado pela ferramenta.

Por fim, foi perguntado aos participantes se eles sentiram alguma dificuldade ou limitação no uso do GitHub Copilot. Seis dos participantes relataram que tiveram alguma dificuldade. Dois deles relataram que o GitHub Copilot tem dificuldades em sugerir respostas corretas para problemas mais complexos. Outros três participantes relataram que as sugestões fornecidas resolviam algum outro problema, ou que se encontrava em um outra classe. Um dos participantes relatou dificuldade no uso da ferramenta, mas que após seguir as instruções enviadas conseguiu contornar facilmente.

6 DISCUSSÃO

A partir dos resultados apresentados pode-se perceber que o Copilot influencia positivamente no percentual de respostas corretas. Do total de respostas recebidas, 65% delas estavam corretas para os envios feitos pelos participantes que tiveram auxilio, enquanto que apenas 43% das respostas enviadas pelos participantes que não tiveram o auxilio estavam corretas.

Outra influência positiva foi no tempo que os participantes levaram para resolver os problemas. Quando compara-se o tempo médio de reposta utilizando o Git Hub Copilot e sem esse auxílio, os problemas resolvidos com o auxílio do GitHub Copilot foram resolvidos em menos da metade do tempo levado para resolver os problemas sem esse auxílio.

6.1 Limitações

Quanto às limitações, este trabalho, que foi um trabalho inicial, apresenta algumas limitações. Limitações essas, relacionadas aos participantes, aos problemas e à linguagem de programação escolhida.

A primeira limitação está relacionada à quantidade de participantes, que foram 20 alunos do terceiro semestre do curso de ciência da computação. Para ter resultados melhores e mais precisos, se faz necessário uma quantidade maior de participantes nesse tipo de experimento, mas infelizmente não foi possível encontrar mais participantes que se encaixassem nas características necessárias para o experimento.

Por ser um experimento inicial os participantes tinham que ser desenvolvedores iniciantes e com pouco experiência. Desse modo, alunos dos primeiros semestres de cursos de computação foram os escolhidos. Uma outra limitação foi o uso do Java como linguagem alvo desse experimento. Então, além de desenvolvedores iniciantes, era necessário que os participantes tivessem conhecimento nessa

linguagem de programação específica, o que limitou ainda mais o número de participantes disponíveis.

Por fim, os problemas escolhidos foram problemas com características de programação estruturada, e não de programação orientada a objetos. Isso foi feito com o objetivo de simplificar e diminuir a dificuldade do problemas, para que eles fossem resolvidos mais rapidamente. Além de lembrar que os participantes conheciam a linguagem Java, mas não tinham tanta experiência com ela.

7 CONSIDERAÇÕES FINAIS

Este trabalho apresentou uma análise do impacto do GitHub Copilot ao auxiliar alunos do curso de ciência da computação em tarefas simples de programação. A partir da análise feita aqui, conclui-se que o GitHub Copilot é uma ferramenta excelente e pode ajudar a reduzir drasticamente o tempo e esforço necessário para resolver tarefas de programação.

A análise estatística mostrou que os dados não seguem a distribuição normal, por isso, foi utilizado o método do Qui Quadrado. Os resultados dessa análise mostraram que os tempos médios dos alunos que utilizaram o GitHub Copilot e os que não utilizaram são estatisticamente significativos, enquanto que para a quantidade de acertos foi encontrada uma diferença estatisticamente significativa.

Acredita-se que essas ferramentas podem ser ótimos guias, auxiliando alunos a não só obter respostas, mas entender melhor a linguagem de programação por meio da visualização de maneiras variadas de resolver o mesmo problema.

7.1 Trabalhos Futuros

Como trabalhos futuros, pretende-se expandir e aprofundar essa análise, aplicando-a em um contexto com participantes mais experientes, no qual eles irão utilizar o GitHub Copilot para auxiliá-los. Para isso, em um primeiro momento serão repassados aos participantes, alguns problemas de orientação a objetos, no quais eles terão de desenvolver sistemas completos utilizando o auxilio do GitHub Copilot. Para esse primeiro momento, os participantes ainda serão alunos, no entanto, serão alunos dos semestres finais de cursos de computação.

Em um segundo momento serão selecionados participantes da indústria, desenvolvedores com algum tempo de experiência na área de desenvolvimento de software. O objetivo será repassar aos participantes, o código fonte de sistemas reais, que estejam em uso. O corpo de alguns métodos serão removidos, então será solicitado aos participantes para corrigirem o problema com o auxilio do GitHub Copilot. O objetivo é verificar a eficiência da ferramenta em um contexto real, se as suas sugestões são realmente úteis.

REFERÊNCIAS

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL]

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 https://arxiv.org/abs/2107.03374

[3] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, Zhen Ming, and Jiang. 2023. GitHub Copilot AI pair programmer: Asset or Liability? arXiv:2206.15331 [cs.SE]

[4] W.W. Daniel. 2000. *Applied Nonparametric Statistics*. Duxbury. https://books.google.com.br/books?id=bCDFAAAACAAJ

[5] Pedro Domingos. 2012. A Few Useful Things to Know about Machine Learning. *Commun. ACM* 55, 10, 78–87.

[6] Neil A. Ernst and Gabriele Bavota. 2022. AI-Driven Development Is Here: Should You Worry? *IEEE Software* 39, 2, 106–110.

[7] GitHub. 2023. Your AI pair programmer. https://github.com/features/copilot. Acessado: 20 de agos. de 2023.

[8] Shudong Huang, Zhao Kang, Zenglin Xu, and Quanhui Liu. 2021. Robust deep k-means: An effective and simple method for data clustering. *Pattern Recognition* 117, 107996.

[9] Mladan Jovanović and Mark Campbell. 2022. Generative Artificial Intelligence: Trends and Prospects. *Computer* 55, 10, 107–112.

[10] M.H. Kutner. 2005. *Applied Linear Statistical Models*. McGraw-Hill Irwin.

[11] Daniel Leiker, Ashley Ricker Gyllen, Ismail Eldesouky, and Mutlu Cukurova. 2023. Generative AI for Learning: Investigating the Potential of Learning Videos with Synthetic Virtual Instructors. In *Artificial Intelligence in Education. Posters and Late Breaking Results, Workshops and Tutorials, Industry and Innovation Tracks, Practitioners, Doctoral Consortium and Blue Sky*, Ning Wang, Genaro Rebolledo-Mendez, Vania Dimitrova, Noboru Matsuda, and Olga C. Santos (Eds.). Springer Nature Switzerland, Cham, 523–529.

[12] Weng Marc Lim, Asanka Gunasekara, Jessica Leigh Pallant, Jason Ian Pallant, and Ekaterina Pechenkina. 2023. Generative AI and the future of education: Ragnarök or reformation? A paradoxical perspective from management educators. *The International Journal of Management Education* 21, 2, 100790.

[13] T.J. McCabe. 1976. A Complexity Measure. *IEEE Transactions on Software Engineering* SE-2, 4, 308–320.

[14] Tom Michael Mitchell. 1997. *Machine Learning*. WCB/McGraw-Hill, Boston, MA. 414 pages.

[15] JunSeong Moon, RaeEun Yang, SoMin Cha, and Seong Baeg Kim. 2023. chatGPT vs Mentor : Programming Language Learning Assistance System for Beginners. In *2023 IEEE 8th International Conference On Software Engineering and Computer Systems (ICSECS)*. 2023 IEEE 8th International Conference On Software Engineering and Computer Systems (ICSECS), Penang, Malaysia, 106–110.

[16] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, and Zhen Ming (Jack) Jiang. 2023. GitHub Copilot AI pair programmer: Asset or Liability? *Journal of Systems and Software* 203, 111734.

[17] Ekaterina A. Moroz, Vladimir O. Grizkevich, and Igor M. Novozhilov. 2022. The Potential of Artificial Intelligence as a Method of Software Developer's Productivity Improvement. In *2022 Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*. 2022 Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus), Saint Petersburg, Russian Federation, 386–390.

[18] Nhan Nguyen and Sarah Nadi. 2022. An Empirical Evaluation of GitHub Copilot's Code Suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories* (Pittsburgh, Pennsylvania) *(MSR '22)*. Association for Computing Machinery, New York, NY, USA, 1–5.

[19] Alberto S. Nuñez-Varela, Héctor G. Pérez-Gonzalez, Francisco E. Martínez-Perez, and Carlos Soubervielle-Montalvo. 2017. Source code metrics: A systematic mapping study. *Journal of Systems and Software* 128, 164–197.

[20] NVIDIA. 2023. Large Language Models Explained. https://www.nvidia.com/en-us/glossary/data-science/large-language-models. Acessado: 20 de agos. de 2023.

[21] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. arXiv:2304.02491 [cs.HC]

[22] Ben Puryear and Gina Sprint. 2022. Github Copilot in the Classroom: Learning to Code with AI Assistance. *J. Comput. Sci. Coll.* 38, 1, 37–47.

[23] Damien Raftery. 2023. Will ChatGPT pass the online quizzes? Adapting an assessment strategy in the age of generative AI. *Irish Journal of Technology Enhanced Learning* 7.

[24] Linda Rosenberg, Ted Hammer, and Jack Shaw. 1998. Software metrics and reliability. In *9th international symposium on software reliability engineering*.

[25] S.J. Russell, P. Norvig, and E. Davis. 2010. *Artificial Intelligence: A Modern Approach*. Prentice Hall. https://books.google.com.br/books?id=8jZBksh-bUMC

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL]
[27] Michel Wermelinger. 2023. Using GitHub Copilot to Solve Simple Programming Problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 172–178.
[28] Han Xue and Yanmin Niu. 2023. Exercise Generation and Student Cognitive Ability Research Based on ChatGPT and Rasch Model. *IEEE Access* 11, 1–1.